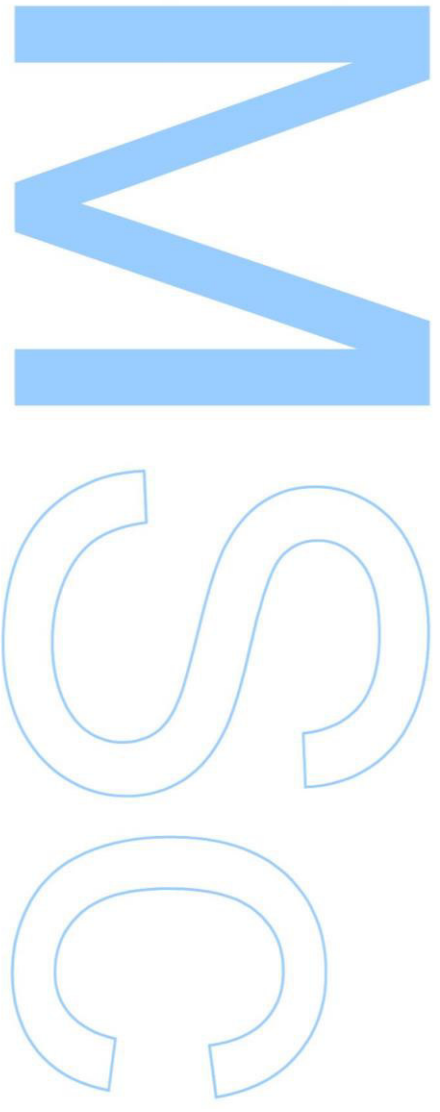
# Ramble: Opportunistic Content Dissemination for Infrastructure-Deprived Environments

Miguel Ângelo Felisberto Garcia
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2018

**Orientador**
Eduardo Marques, Professor Auxiliar Convidado, DCC/FCUP

**Coorientador**
Luís Lopes, Professor Associado, DCC/FCUP




Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

# Abstract

Mobile devices have become ubiquitous. As time progresses, these devices have become more powerful, having larger computational and memory capacities. Besides these advances, the majority of smartphones in the market today include Device-to-Device (D2D) communication technologies such as WiFi-Direct and Bluetooth. Despite being present in a large portion of mobile devices, these technologies are not always used to their full potential. Smartphones have also become a great tool for crowdsourcing, thanks to their large number and input capabilities.

These factors have given rise to a field of study that focuses on connecting nearby devices in order to pool their resources together to perform, for example, distributed computation. This work focuses on the scenario where there is no infrastructure support, such as a disaster one. In these situations, a system that provides content dissemination through the use of mobile devices and cloudlets could prove useful, as users could generate and share data that would help rescue teams.

In this context we propose Ramble, an opportunistic content dissemination system for mobile devices and cloudlets. Ramble is designed around using as many contact opportunities as possible to make a connection and synchronise with other peers. These contacts can be made using WiFi, bridging mobile devices with cloudlets and with each other, WiFi-Direct, bridging two or more mobile devices nearby by establishing a temporary network between them, and by using a mesh network, connecting cloudlets together allowing the spread of information over larger distances.

For our case study we chose to use the system to disseminate geotagged content generated by mobile devices, such as audio and video recordings, pictures and written reports. Around this idea, we developed an Android application that allows users to view content stored in the device and generate new content. We also developed a cloudlet application that functions as a drop-off point, gathering content as users pass by.

We performed a real world experiment in order to validate Ramble, derive preliminary metrics and identify current problems.

# Resumo

Os dispositivos moveis tornaram-se ubíquos. Com o passar do tempo, estes dispositivos tornaram-se mais poderosos, tendo maiores capacidades computacionais e de armazenamento. Além destes avanços, a maioria dos smartphones no mercado atual incluem tecnologias de comunicação dispositivo-para-dispositivo, como WiFi-Direct e Bluetooth. Apesar de estas estarem presente em grande parte dos dispositivos móveis, nem sempre são utilizadas em todo o seu potencial. Os smartphones também se tornaram uma ótima ferramenta de crowdsourcing, graças ao seu grande número e vários inputs.

Estes fatores deram origem a um área de estudo que se baseia em usar dispositivos moveis na vizinhança a fim de reunir os seus recursos para executar tarefas como, por exemplo, computação distribuída. Este trabalho concentra-se no cenário em que há falha de infraestruturas de comunicação, como um desastre. Nessas situações, um sistema que forneca disseminação de conteúdo usando dispositivos móveis e cloudlets pode ser útil, pois os utilizadores podem gerar e compartilhar dados que ajudariam as equipas de resgate.

Neste contexto, propomos o Ramble, um sistema oportunista de disseminação de conteúdos para dispositivos móveis e cloudlets. Ramble é desenhado em torno de usar todas as oportunidades de contato possiveis para estabelecer uma conexão e sincronizar com outros nós. Esses contatos podem ser feitos usando WiFi, conectando dispositivos móveis entre si ou com cloudlets, WiFi-Direct, conectando dois ou mais dispositivos móveis próximos, estabelecendo uma rede temporária entre eles e usando uma rede mesh, conectando as cloudlets entre si, permitindo a disseminação de informações em distâncias maiores.

Para o nosso caso de estudo, optamos por usar o sistema para disseminar conteúdo georreferenciado gerado por dispositivos móveis, como gravações de áudio e vídeo, imagens e relatórios escritos. Em torno desta ideia, desenvolvemos uma aplicação Android que permite aos utilizadores visualizar o conteúdo armazenado no seu ispositivo e gerar novos conteúdos. Também desenvolvemos uma aplicação para cloudlets que funciona como um ponto de entrega, reunindo conteúdo à medida que os utilizadores passam.

Realizamos uma experiência de campo para validar o Ramble, derivar métricas

preliminares e identificar problemas atuais.

# Acknowledgements

Firstly, I would like to thank my advisers, Eduardo Marques and Luís Lopes, for their leadership, patience and assistance throughout the dissertation. I also thank all the people who helped me in this work, either by participating in the experiment or giving me advice.

I would also like to thank my friends for their support and camaraderie, for being there along the way.

Last, but definitely not least, I thank my parents for their unwavering support and belief. Without them none of this would be possible and I'm eternally grateful for it.

# Contents

# List of Tables

# List of Figures

# Listings

# Acronyms

**API** Application Programming Interface

**AP** Access Point

**BATMAN** Better Approach To Mobile Ad-hoc Networking

**D2D** Device-to-Device

**DTN** Delay-Tolerant Network

**GO** Group Owner

**GPS** Global Positioning System

**gRPC** gRPC Remote Procedure Calls

**GSM** Global System for Mobile Communications

**IDL** Interface Definition Language

**IP** Internet Protocol

**IoT** Internet of Things

**JDBC** Java Database Connectivity

**LAN** Local Area Network

**MANET** Mobile Ad-hoc Network

**MCC** Mobile Cloud Computing

**OLSR** Optimised Link State Routing

**OSM** OpenStreetMap

**OS** Operating System

**P2P** Peer-to-Peer

**RPC** Remote Procedure Call

**SMS** Short Message Service

**SPAN** Smartphone Ad-hoc Network

**SSID** Service Set Identifier

**SaaS** Software as a Service

**TDLS** Tunnelled Direct Link Setup

**UDP** User Datagram Protocol

**UUID** Universally Unique Identifier

**VANET** Vehicular Ad-hoc Network

**WMN** Wireless Mesh Network

# Chapter 1

# Introduction

## 1.1 Motivation

Mobile phone usage has grown in recent years. It is estimated that the number of smartphones will reach 2.53 billion in 2018, a number that is expected to continue growing in the following years, going as high as five billion by 2019 [1]. This increase in users is largely caused by the lowering cost of manufacturing devices. Smartphones aren't merely growing in users, but also in processing power, capacity and battery life, to name a few. Today's devices are able to perform computationally intensive tasks, such as rendering 3D graphics in games or playing high-quality video. Besides these types of tasks, smartphones have also become a source for data collection, for instance taking pictures, recording video and sound, reading fingerprints and performing geolocation. There's also been advances in the inclusion of wireless Device-to-Device (D2D) communication technologies such as WiFi-Direct, WiFi Tunnelled Direct Link Setup (TDLS) and Bluetooth along with the traditional WiFi and 3G/4G.

These qualities greatly empower crowdsourcing applications, where the information is generated in a bottom-up fashion. In these tools, the data is generated and collected by the users and incorporated into the main system. Crowdsourcing tools have been used in aiding disaster recovery in some scenarios [2, 3] due to the main advantage of crowdsourced information, data volume. Users already on the site can collect information more quickly and detailed than rescue personnel, though it may need to be verified either by a trusted organisation or put through a crowd verification mechanism, such as voting. An example of that would be the dissemination of geotagged announcements, similar to the service provided by Scipionus Map [4], a simple web service that allows users to place and view markers on a Google Map.

## 1.2 Problem statement

The problem with cloud-based services arises when connectivity to the cloud can't be guaranteed. In the event of infrastructure failure, many of the existing services cannot be accessed. Natural disasters, for instance large hurricanes and earthquakes, are known to disrupt communications by physically damaging the network structure due to the inherent destruction caused by such events. An architecture that makes use of cloudlets, thin servers close to the edge of the network (i.e., close to the location of users) that provide services, and D2D connectivity to form networks composed by mobile devices could greatly help to keep some of these services functioning.

Disasters also generate unusually high traffic that can severely limit the ability to communicate, as the network becomes congested. In the Great East Japan Earthquake, a survey was conducted to assess the communications following the earthquake [5], which showed very low degree of satisfaction in the days immediately after the event.

By using cloudlets or devices in the neighbourhood, data generated by one device can be transferred to others and vice-versa, providing updated information and allowing data to spread. Examples of other applications that would benefit from such a system would be sharing voice messages, either directly or through a publish-subscribe architecture and the collection of forms to more easily generate reports of the structured data.

To make a service run in the event of infrastructure failure and to take advantage of crowdsourced data, we propose a system that makes use of mobile devices and cloudlets to disseminate data. The exchange of data between devices would be done opportunistically, taking advantage of various contacts eventually established between devices. Mobile devices can form temporary networks with other devices or connect to nearby cloudlets. Both of them can connect directly to the cloud, if available. The shared data can be of any kind, but in this kind of scenario the sharing of geotagged reports, text, audio and video messages, news and map data would make sense to both the users and, for example, rescue teams. Other kinds of information could be sensor data.

## 1.3 Contributions

The main contributions of this work are the following:

- We designed a data sharing library that is able to discover other peers with the goal of disseminating contents. The system manages content storage, content metadata, peers, subscriptions, peer discovery, message exchanges and filter management;

- Implemented the system targeted at mobile devices and cloudlets;
- Designed and implemented an Android app that allows end users to easily view and generate content whilst using the system;
- Implemented, using the Link Layer of the Hyrax Middleware, a routine to form WiFi-Direct networks between nearby devices aimed at short-term opportunistic contacts;
- Performed a real world test in order to validate, detect problems and extract preliminary metrics of the system that can potentially lead to future improvements.

## 1.4 Thesis Structure

The thesis is structured as follows. Chapter 2 contains key concepts and technologies in which this thesis is based upon. In Chapter 3 we present the state-of-the-art relevant to this work, including previous work developed in the project, existing communication frameworks and cloud services. Chapter 4 details the design and implementation of the Ramble system, describing the various components and how they were implemented. Chapter 5 describes and displays the results of the experiment that we performed in order to provide proof of concept. Finally, Chapter 6 provides a general overview of this dissertation along with future improvements.

# Chapter 2

# Background

This chapter presents concepts and technologies in which this thesis is built upon. Section 2.1 summarises the three main service paradigms we use in this dissertation, section 2.2 describes the WiFi-Direct technology, outlining its main characteristics and procedures and section 2.3 references the Hyrax project, defining its main goals and case studies.

## 2.1 Cloud Computing, Cloudlets and Mobile Edge-Clouds

To place this work in context we first take a look at cloud computing, which is a computing paradigm where resources such as storage and computation are provided over the internet. The infrastructure that provides such resources can be called a cloud. Cloud resources are typically pooled to serve multiple consumers. Resources are allocated and freed dynamically according to user needs [6].

Smartphones make great use of this paradigm, as they are computationally limited and must be energy aware. Therefore, many smartphone applications adopt the Software as a Service (SaaS) service model, where the client makes use of an application running on the cloud through a thin client interface, such as a web browser or a program, without having control of the underlying cloud infrastructure.

Although clouds cover a wide range of use cases, they can fall short in applications requiring low latency and high bandwidth, as they are usually housed in data centres that can be physically distant. There is also the obvious fact that clouds become inaccessible in the case of communication failures.

To aid in this scenario, a mid-level element called *cloudlet* can be deployed to reduce

the distance to the cloud. Cloudlets can be used as a closer and smaller cloud that acts on the its behalf, performing computation or serving as caches. Because they are placed at the edge of the network, near end-users, they can potentially reduce end-to-end latency and low bandwidth problems. When provided with enough hardware, cloudlets can enable a new set of applications that clouds could not, such as wearable cognitive assistance [7].

Besides the goal of reducing latency, cloudlets have uses in the event of communications or cloud failure. Since cloudlets act as surrogates of the cloud, they can transparently mask its absence, providing a limited version of its services [8].

Finally, we have ad-hoc networks. These networks are decentralised networks formed by wireless links. Each node in the network can route traffic, a task that in traditional networks is done by routers. A specific kind of ad-hoc network is the Mobile Ad-hoc Network (MANET). MANETs are ad-hoc networks built by mobile devices. One particular characteristic of MANETs is that the nodes forming the network are highly mobile, constantly changing links with other devices, which hinders routing. This means that a MANET must be self-configuring as to allow data to flow correctly through the network.

Examples of MANETs are Smartphone Ad-hoc Networks (SPANs), networks built with smartphones that use the communication interfaces present in those devices to form the network, and Vehicular Ad-hoc Networks (VANETs), for communication between vehicles and fixed stations.

## 2.2 WiFi-Direct

WiFi-Direct/WiFi P2P [9] is a communication technology that enables devices to communicate directly without the need of an Access Point (AP). Unlike ad-hoc mode, WiFi-Direct is built on top of the IEEE 802.11 infrastructure mode, in which devices either act as APs, connected to a Local Area Network (LAN), or clients [10]. In WiFi-Direct, a device that provides AP functionality in a group is called a P2P Group Owner (GO) and a client is called P2P Client, from now on simply referred to as GO and Peer. These roles are dynamic and can even be implemented simultaneously by the same device at the same time (on multiple physical interfaces).

To form a connection, a device must first discover other WiFi-Direct devices. That's accomplished by an initial scan phase, to find existing networks or already formed WiFi-Direct groups, followed by the search and listen phases, in which the discovering device sends Probe Requests (active scanning) and then listens for them, respectively. These two states are variable in duration and alternate until the discovering process stops. In this process, extra data can be exchanged in order to provide information about the services

available on the device being discovered, prior to establishing a connection. This is called Service Discovery.

In the absence of a previously formed group between the scanning and the discovered device, the GO negotiation process is then performed. This consists on a three way handshake in which both devices agree on certain parameters. In this handshake, a Group Owner Intent is exchanged. The GO is the device with the highest value.

Finally, WPS Provisioning is performed in order to create a secure connection, along with DHCP address assignment.

What gives potential to this technology is using already implemented and widely used mechanisms. Today WiFi-Direct technology is used in printers and TVs by services such as Miracast.

WiFi-Direct is also used to empower Peer-to-Peer (P2P) applications such as the dissemination of video content between Android devices in close proximity [11, 12], as the high bandwidth and low latency is suitable for transferring large files without requiring the use of the network infrastructure, such as APs.

One major drawback lies on the fact that WiFi-Direct only provides single-hop communication between devices. Two devices connected to the same GO can communicate by using the GO as gateway, even though they are in range of each other. This GO mechanisms also hinders the usage of WiFi-Direct for forming ad-hoc networks, as devices would need multiple interfaces performing different roles to maintain connection.

## 2.3 The Hyrax Project

Hyrax [13] is a project aimed at facilitating the development of edge-cloud applications. To that end, a middleware was developed with the main goal of abstracting away all the intricacies of using various communication protocols in a simple to use API to form MANET without requiring root access. Hyrax is motivated by three main scenarios.

User-generated Replays [11, 12] is an application that allows users in a crowded venue, such as a sports match or a race, to capture and share video with other users in the vicinity. In such events, it is commonplace that infrastructural access, namely WiFi and mobile communications, becomes congested, due to the high density of devices present in a certain location. To mitigate this, devices could share video content locally by means of an edge-cloud, relieving infrastructure of high traffic. A practical test was performed that showed not only an increased number of users served by the edge-cloud, but also high speeds when compared with video download through an AP.

Another considered scenario is distributed face recognition [14]. This application can be useful when dealing with a missing person in a crowded place. By using edge-clouds, face recognition can be performed locally on each device using the pictures stored there, without having to disclose them to a cloud service.

This work focuses on this last scenario, which is providing connectivity in case of prolongued infrastructure deprivation.

### 2.3.1 The Hyrax Middleware

The middleware, as previously stated, enables the formation of edge-clouds for application development. The architecture of the middleware is composed by the link layer, the network layer and the service layer.

The link layer is the bottom layer and it handles the creation of links between devices using the different communication technologies Hyrax supports. Currently, Hyrax supports WiFi, WiFi-Direct, Bluetooth, Bluetooth LE and 3G/4G. A simple and unified Application Programming Interface (API) is provided to use all technologies for discovery and connection to other devices. We make use of the Link Layer in this work to perform P2P discovery and group formation. This topic is discussed in chapter 4.3.2.2.

The network layer is built on top of the link layer and it is responsible for establishing a logical network abstraction using the links set up using the link layer, handling formation, logical address translation and routing. The API provides methods for packet and stream based communications, such as sending data to a specific peer or broadcast.

Lastly, the service layer encompasses core services to be used by applications, such as distributed storage, parallel computing and publish-subscribe systems.

# Chapter 3

# State of the Art

In this chapter we present previous work and projects relevant to this dissertation. In section 3.1 we present a simple messaging application that was developed in the scope of the Hyrax project. In section 3.2, we take a look at various projects that revolve around Peer-to-Peer (P2P) communications, providing a comparison between them. Finally, in section 3.3 we present cloud services that have been used in previous disaster situations.

## 3.1 HyraxMsg

Relating off-grid communications, a messaging application was developed in the Hyrax Project. HyraxMsg [15] is an Android application that provides users in the near vicinity with a medium to exchange text messages similar to FireChat. Communication between users can be made using WiFi, WiFi-Direct and Bluetooth, conveniently provided by the Hyrax Middleware, and User Datagram Protocol (UDP) sockets. The application was built as a use case for the middleware to assess the formation of the network with a practical case.

The application provides a simple user interface with a list of contacts representing users in the edge-cloud. That functionality is provided by the Network Layer of the middleware. When a contact is selected, the user can then exchange text messages with it and know its Global Positioning System (GPS) coordinates.

## 3.2 Communication frameworks

In this chapter we highlight some platforms with Mobile Ad-hoc Network (MANET) capacities that are related with this project.

### 3.2.1 The Serval Project

The Serval Mesh [16] is an application for Android that provides useful features in the event of a disaster or communication failure, such as text messaging, file sharing, voice calls and map tile sourcing. The system is capable of creating self-organising mesh networks (MANETs) with mobile devices using ad-hoc connections. Communication can also be made using Bluetooth and WiFi, by creating an Access Point (AP) or connecting to one. Serval are also developing the Serval Mesh Extender, a device that acts as an access point for nearby devices and that connects to more distant extenders using UHF packet radio. A major drawback of Serval is the need for super-user permissions (or rooting) on the devices. Although it allows for the AP, client and Bluetooth modes, ad-hoc is restricted to rooted devices. This is due to Android not providing ad-hoc mode by default through the Application Programming Interface (API), though it provides WiFi-Direct and Tunnelled Direct Link Setup (TDLS), which Serval does not seem to employ to circumvent this limitation. Serval software and hardware has been subject of some field tests, some of which are range tests for the extenders. Testes in an urban scenario showed reasonable connection in a distance of about 250m with extenders placed inside houses. The project is being developed in Flinders University, Australia and its completely open-source.

### 3.2.2 FireChat and MeshKit

FireChat [17] is a messaging application developed by Open Garden that uses both infrastructure and MANETs to allow users to communicate, making use of both Bluetooth and WiFi. FireChat is built using the MeshKit SDK [18], a library that allows the integration of mesh network in application. The SDK supports various wireless protocols like Bluetooth, Bluetooth LE and WiFi Direct and allows for seamless use of both Device-to-Device (D2D) and cloud access. This however couldn't be tested as the SDK is closed-source and only available to "select organisations" such as media companies, telecommunication operators and NGOs. MeshKit has undergone a few practical tests, namely in a national convention, a music festival and an earthquake preparedness simulation in the Philippines. This latter one displayed that MeshKit allowed information to reach 80% more people when compared to cellular connections, in a density of 700 $users/m^2$.

### 3.2.3 Commotion Wireless

Commotion Wireless [19], unlike the previous examples, targets the infrastructure side of Wireless Mesh Networks (WMNs). To that end, Commotion developed software to be integrated in wireless routers that builds upon existing open-source projects, for example

OLSR, a link-state routing protocol for mesh networks, OpenWRT, a Linux distribution for embedded devices commonly used to deploy routers and Serval. The goal is to provide a decentralised infrastructure for communication. Commotion was implemented in a Tunisian town named Sadaya where it serves as a host for services like OpenStreetMap, Wikipedia, free digital book library, file sharing and Etherpad, a platform for collaborative document editing.

### 3.2.4 goTenna

goTenna's [20] approach to creating a MANET differs from the ones presented previously. Instead of using the antennas of mobile devices, communication is made with a portable device. This device is lightweight, battery powered and can be paired with smartphones through Bluetooth to add mesh networking to apps. This allows for chatting applications and GPS coordinates sharing. Communication between goTenna devices is made using radio frequencies of 151-154 MHz, allowing for greater distances and obstacles, however these are highly dependent on terrain conditions. One advantage of this approach is the use of meshing, which allows for users to communicate through greater distances than they would be able to using a direct link. A limiting factor is the fact that goTenna devices can only interact among themselves. The device is sold in pairs and is targeted to groups of people that venture into locations without connection, such as remote villages or mountains, but it could have great use in a disaster scenario, though the device is a bit expensive for a radio antenna.

### 3.2.5 Uepaa p2pkit

p2pkit [21] is a library with a simple API that allows smartphones to connect and estimate the distance to other devices in near proximity. Like Hyrax, the library helps abstract the complicated APIs that implement the various communication technologies, such as Bluetooth and WiFi. Unlike Hyrax, p2pkit does not implement any form of routing for allowing multi-hop communication by default. The API, whilst simple, seems to be more limiting than general purpose mesh networking frameworks, containing APIs for Discovery, to manage the discoverability of devices, Lifecycle, to manage Uepaa account validation, Messaging, to send and receive messages and Proximity Ranging, to estimate the relative distance between two devices.

### 3.2.6 AllJoyn

AllJoyn [22] is an open-source framework that allows devices to communicate using D2D connections. The platform is Operating System (OS) agnostic, supporting all major smartphone and Internet of Things (IoT) OSs. The framework works by allowing devices to publish APIs over the network representing the functionalities a certain device offers. AllJoyn handles the discovery of these APIs over many communication technologies. This is achieved by means of a general bus, which abstracts all discovery mechanisms of the various communication technologies. These include WiFi, WiFi-Direct, Bluetooth and Ethernet.

### 3.2.7 Comparison

Table 3.1 shows the different frameworks considered in chapter 3.2 resuming some of the common characteristics, denoting whether:

- they are open-source,
- they support device mobility,
- they implement MANET or mesh capabilities,
- they support WiFi, WiFi-Direct and Bluetooth.

Table 3.1: Table highlighting the differences between the previously described frameworks and services.

| Framework | Open-source | Mobilily | MANET or mesh | WiFi | WiFi-Direct | Bluetooth | Observations |
|---|---|---|---|---|---|---|---|
| **The Serval Project** | Yes | Yes | Yes* | Yes | No | Yes | *Requires root |
| **MeshKit** | No | Yes | Yes | Assumed | No | Yes | |
| **CommotionWireless** | Yes | No | Yes | Yes | No | No | |
| **goTenna** | No | Yes | Yes | No | No | No | Meshing requires a specific version of the portable antenna |
| **p2pkit** | No | Yes | No | No | Yes | Yes | Requires API key |
| **AllJoyn** | Yes | No | Yes | Yes | Yes | Yes | |
| **Hyrax Middleware** | Yes | Yes | Yes | Yes | Yes | Yes | |

## 3.3 Cloud services used in disaster scenarios

We highlight some services known to be used in disaster relief.

### 3.3.1 FrontlineSMS

FrontlineSMS [23] is an open-source software that allows for the use of mobile devices or GSM modems to distribute and collect information by using the SMS capabilities of those devices. It has the advantage of connecting a large number of users without the need for an internet connection, requiring only GSM.

### 3.3.2 OpenStreetMap

OpenStreetMap (OSM) [24] is a crowdsourced map service. Map data is generated by users following a collaborative model similar to Wikipedia [25]. The web interface provided is similar to Google or Bing maps, where the users can zoom, pan and search for a location. Additionally, OSM also provides an export function, to allow users to download map data in various formats, an editing function, for submitting changes or uploading GPX traces, and a community wiki, with guidelines on how to collect and submit map data.

This functionality proved useful in the Haiti Earthquake [2] as the map of many haitian locations, namely the capital Port-au-Prince, were updated to match the current state of the city.

### 3.3.3 Ushahidi

Ushahidi [26] is a platform that makes use of crowdsourcing to provide a version of events that is closer to reality. If we consider that in some places media can be censored, the population can then contribute with information, that is verified by the crowd [27]. The platform allows for anonymity and the use of SMS, MMS and a browser, which makes it versatile and accessible. Besides being used in censored media environments, it proved convenient during the Haiti Earthquake. By gathering geotagged reports from users, a more accurate perspective of events in a certain location can be formed [2].

### 3.3.4 Zello

In scenarios of natural disasters many voice and text messaging applications were used to provide information and communication to people. One example of a widely used one is Zello [28], a walkie-talkie application that provides a simple service that was very useful during the Irma Hurricane [29]. The application has a push-to-talk function, where users can record voice clips and either send them directly or broadcast them to a group of other users. This was used during said hurricane to help rescue teams find people in need, as

many houses were completely submerged and there was risk of drowning. People used Zello to broadcast their location to rescue teams as well as receive and send information. We highlight this particular application due to its simple nature and high utility. Zello uses does not provide any form of D2D connectivity, which we think it could highly benefit from.

### 3.3.5 iGDACS and Natural Disaster Monitor

iGDACS and Natural Disaster Monitor are two smartphone apps targetting iOS and Android, respectively, that provide real-time information about natural disasters and gives the possibility to send back information in the form of a geo-located image and/or text. iGDACS is the official app by the European Commission, while Natural Disaster Monitor uses the Global Disaster Alert and Coordination System (GDACS) website for information.

# Chapter 4

# Design and Implementation

As was briefly introduced in chapter 1, Ramble aims at being a system that takes advantage of edge components and user mobility to disseminate content opportunistically. Although it was designed in such a way that allows many different types of devices to use it, we focus on two main types: mobile devices and cloudlets. Both these components, although different in goal, share the same underlying software. They store information and discover other peers on a local network using the same methods. Mobile devices, however, need an extra discovery mechanism in order to use WiFi-Direct. Despite that, cloudlets, for example, interact with mobile devices the same way they interact with other cloudlets, and vice-versa. Instead of designing a different software stack for each component, we use the same ones with different configurations. The system uses a client-server methodology, in which each device acts as both a client and a server. A remote server component could also be added, with both mobile devices and the cloudlets periodically checking for a connection to it, although the same message dynamic could not be used. These components and interactions are explained in this chapter. Section 4.1 defines the architecture, outlining the different components and what types of connections they use to reach each other. Section 4.2 formally describes the structure of the data used by in the database and in the exchange messages. Finally, section 4.3 defines the software components that are used in the components.

## 4.1 Architecture

Ramble was mainly designed to be used by three types of physical devices, which we will refer to as Mobile devices, Cloudlets and Cloud components.

In this work, we considered mobile devices to be WiFi and WiFi-Direct enabled Android devices. This, for the most part, was because the current implementation of the Hyrax

Link Layer is an Android Library as the other main mobile operating system, iOS, does not support direct use of WiFi-Direct. Theoretically, the same system could be implemented in another mobile operating system given the same conditions. Mobile devices are essential to this work as they provide the mobility that Ramble takes advantage of. They also provide a means for a user to interact with the system. As mentioned before, mobile devices are found everywhere today, which also empowers the use Ramble.

Next, we define Cloudlets. Although these devices are typically powerful servers used to offload computation, we use them to offload storage. With this, cloudlets need not be powerful, but instead have large storage capacity to hold large quantities of data. Raspberry Pis make good cloudlets in this scenario, as they are cheap and provide inputs for storage capacity expansion. Cloudlets also provide WiFi access for mobile devices, giving them access the cloudlet but also providing a means to communicate with each other. As an extension to the functionality described before, cloudlets can also be equipped with a second radio to be used in ad-hoc mode paired with a routing protocol (like Better Approach To Mobile Ad-hoc Networking (BATMAN)) to communicate with other cloudlets in the vicinity forming a mesh network between them. This has the effect of keeping the cloudlets synchronised and making information available over large distances.

Finally, the Cloud component is an additional (but non essential) component of the system. A remote server provides the final glue in all the system, collecting information provided by the other components to be processed and analysed centrally.

## 4.2 Data Model

In this section we present the data model that supports all the components in the system.

The data model has three main components, the *Peers*, the *Content* and the *Subscriptions*.

A *Peer* represents a node in the system and is identified by a Universally Unique Identifier (UUID). We use randomly generated (type 4) UUIDs because there's a very low chance that the same UUID is generated in different devices, and also due to the fact that these can be generated independently, without relying on a centralised party. Besides the UUID, more information about a *Peer* could be added. We include the username and a device type, to ease human visualisation, as well as the version of the peer when it was last seen. Internally, only the UUID is used for identifying a peer. Although a UUID is 128 bits in size, we used the larger (37 bytes) string representation to facilitate the implementation.

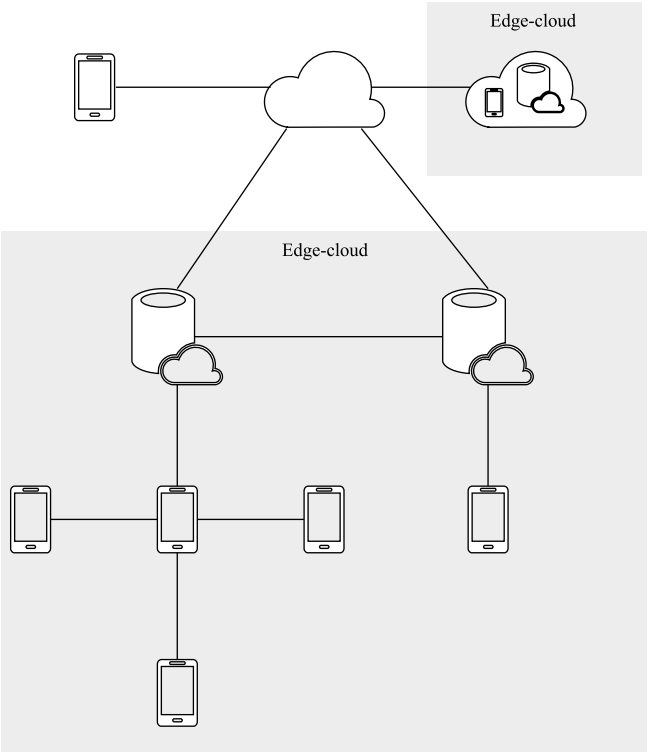



Figure 4.1: Diagram showing the different communication opportunities available in the system.

Next, we define the contents that constitute the data exchanged by devices. We don't impose a restriction on specific types of contents, so a *Content* is some collection of bytes described by a *ContentMetadata*. This holds all the information that describes a content, such as name, creator, size, when and where it was generated and a MD5 hash. Part of the *ContentMetadata* are also content types, which can be audio, video, image, report or other. Lastly, each *Content* contains a list of tags, which can help to further differentiate various kinds of data.

Finally, we define a *Subscription*. Subscriptions are a set of restrictions applied in the moment of device synchronisation. A device/user can only be interested in a subset of all the contents, like pictures taken 1 hour ago in a 5 km radius, to give an example. The content can be filtered by type, by tags, by the peer that created it, by the timestamp of creation, location and size.

These basic data definitions are used in various components, described in this chapter.

## 4.3 Software Components

The Ramble system is divided into logical components, each one with a defined task. The Database is the base component, serving the purpose of storing information about an instance of the system (the state of a device at a point in time). The Discovery Service handles the discovery of other peers in order to establish a connection between them. The Message Service handles message exchanges between discovered peers using a defined protocol. Finally the Android, Cloudlet and Cloud apps that use these services to run the system.

All the components are implemented in the Java programming language, which makes it easy to use the same code across all components. The database, discovery and message service (described later in this section) are the same in both the cloudlet and mobile device components, but have different configuration parameters when deployed on one or the other. This section describes each software component used in the system, outlining what they are, what their function is and how they were implemented.

### 4.3.1 Database

The database is one of the most important components, as all the other ones use it in one way or another. The schema was designed directly from the data model. A diagram of the schema is displayed in figure 4.2.

The *peers* table has the same fields as the ones described in the model. The *subscriptions* and the *content_metadata* tables are also straightforward, each one containing two additional tables to represent multiple fields. The *tags_subscription* and *types_subscription* stores the tags and types of the subscriptions to be sent and filtered. The *tags_content* stores the various tags of the files. The database is, however, not normalised due to the *file_content* table existing, which was a result of a previous state in development. This table should be removed and replaced by a *filename* field in the *content_metadata* table.

#### Implementation

The database is stored in disk using the SQLite[1] database engine. One of the reasons for picking SQLite is that, unlike traditional database management systems which run on a server, it allows an application to perform direct reads and writes to a file using a library. In order to use the the same code in all components, we decided to use Java

[1] https://www.sqlite.org/

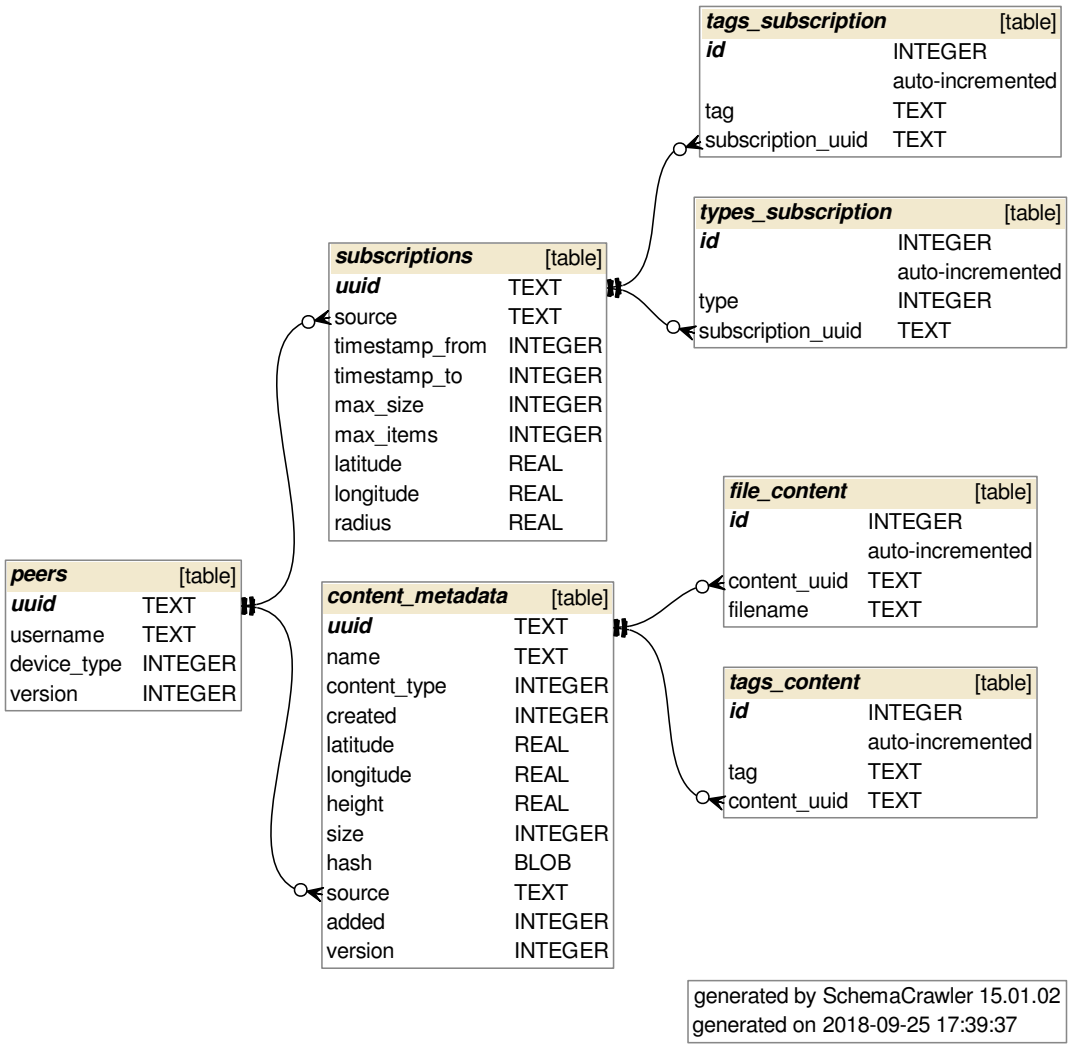



Figure 4.2: Schema of the database.

Database Connectivity (JDBC). JDBC is an Application Programming Interface (API) of the Java Standard Library used to interact with relational databases. One advantage of using JDBC is its Driver Manager. An application can use the same code to make queries and updates, but the underlying driver performs the operations to the specific database. We made use of this feature by supplying different drivers when running on Android and when running in the cloudlets. The Android library already has APIs to deal with SQLite databases, so we used driver[2] that wraps around them. The driver used in the other components (running Linux) wraps around the standard Java SQLite API[3].

[2]https://github.com/SQLDroid/SQLDroid
[3]https://github.com/xerial/sqlite-jdbc

### 4.3.2 Discovery Service

For components to interact with each other, a discovery system needs to exist. The discovery job in Ramble is described in the following subsections and is twofold:

1. Discover peers running Ramble in the same network/subnet, which we will call Service Discovery.
2. Discover other mobile devices nearby in order to establish a WiFi-Direct connection, which we will call Peer-to-Peer (P2P) Discovery.

#### 4.3.2.1 Service Discovery

In case 1 we opted to use a bare-bones broadcast-based discovery mechanism as it fits all requirements and allows us to use the same implementation across the Mobile Device and Cloudlet components. The broadcast has the UUID and version of the peer. Upon finding a peer, the Discovery Service informs the Message Service of the new peer.

**Implementation**

This procedure was implemented by sending a User Datagram Protocol (UDP) packet to the broadcast address, which in IPv4 consists of setting the host portion of the Internet Protocol (IP) address (given by the subnet mask) to ones, of all network interfaces on a specific port. At the same time, the device also listens for UDP packets on the same port. When a packet is received, the sender address is used in order to communicate with it. This, along with the contents of the packet, is passed to the Message service.

#### 4.3.2.2 P2P Discovery and Group Formation

This section describes the process of finding nearby mobile devices and forming a network between them so that they can communicate. The added challenge of doing it in an opportunistic scenario is that we can't guarantee that the network will stabilise. The scenario for which Ramble was designed for is one that presupposes constant movement of mobile devices and contacts between devices are short and must be taken advantage of. It is also important to mention that, at this point, we didn't take energy or security concerns into consideration.

In Ramble we decided to use WiFi-Direct, as it is available in a large portion of devices and provides high speed transfers, which can prove advantageous when transferring large

files. Due to the nature of WiFi-Direct, the trivial solution is to adopt a star-shaped network topology, in which the centre of the star is the Group Owner. As was stated in section 2.2, WiFi-Direct by itself doesn't allow the formation of a mesh network, so in order to take advantage of every contact, we built upon the basic WiFi-Direct Group Formation. Another feature that WiFi-Direct is lacking is the ability to transfer the role of Group Owner to other devices. This can be achieved by destroying a formed network and having the other device create a new one where he is the Group Owner. With this in mind, we implemented a procedure that tries to avoid missed connections.

The procedure for finding and connecting to peers is described in pseudocode in procedure 1. It starts by verifying if the device is connected or is awaiting a connection result, in both cases stopping the process. Then proceeds to check if the device is a Group Owner and has no connected peers, killing the group. It then proceeds with a scanning process, searching for nearby broadcasting WiFi-Direct devices running the Ramble app. All devices are comparable, meaning they can be ordered. Upon finishing the scanning process, discovered peers are ordered in descending order, placing highest one in the first position. If the discovering device is a Group Owner it ends the process. To pick a connection, first we check if there are Group Owners in the discovered peers, connecting to the first one. If no Group Owners are found, one of two situations can happen. If the discovering device is the highest one, it does nothing (it waits for the other to connect), otherwise it connects to the first one.

#### Implementation

The procedure described above was implemented using the Link Layer of the Hyrax middleware. The Link Layer is a library that allows access to the various communication technologies present in Android devices, such as WiFi-Direct and Bluetooth LE. The API provided by the library is unified across all technologies and contains methods, among others, for enabling an interface, making it visible, start discovery and connect to a device, as well as the reverse of those (disable, invisible, stop discovery and disconnect). These methods can be configured based on the communication technology being used and require a listener that the library uses to inform the program about events. We make use of these methods to build our procedure in an asynchronous manner. The Link Layer is also used to search for Access Points (APs) broadcasting a specific Service Set Identifier (SSID) which identifies the cloudlets. Upon finding one, the device connects to it.

**Procedure 1** Procedure for finding and connecting to peers using WiFi-Direct

**Input:** myDevice, linkLayer

```
if linkLayer.isPeer() or awaitingConnection() then
    return
end if
if linkLayer.isGroupOwner() and linkLayer.connectedDevices().isEmpty() then
    linkLayer.killGroup()
end if
devicesFound ← linkLayer.scan()
if linkLayer.isGroupOwner() then
    return
end if
devicesFound ← sortById(devicesFound)
groupOwners ← filterGroupOwners(devicesFound)
devices ← filterDevices(devicesFound)
if linkLayer.isGroupOwner() then
    return
end if
if groupOwners.isNotEmpty() then
    linkLayer.connect(groupOwners[0])
    return
end if
if devices.isNotEmpty() then
    targetDevice ← devices[0]
    if myDevice > targetDevice then
        setAwaitingConnection()
        return
    else
        linkLayer.connect(targetDevice)
    end if
end if
```

### 4.3.3 Message Service

The message service handles incoming and outgoing messages between peers. It sits on top of the discovery service which informs it of found peers. Messages are handled in a client-server dynamic, in which one peer (client) makes a request to another peer (server) and this one returns the response. The messages that we used were *hello* and *pull*. The *hello* message functions as a greeting between the peers, where a peer sends his subscription

and receives a list of contents that match that subscription. The *pull* message is used to obtain a content from another device.

A typical sequence of messages between two peers starts with one sending an *hello* message, followed by zero or more *pull* messages, according to the required content. The other peer mirrors this interaction. At the end of this interaction, both devices are synchronised.

### Implementation

The message service was implemented using a combination of Google's Protocol Buffers and gRPC Remote Procedure Calls (gRPC) frameworks. Protocol Buffers (Protobuf) is used for serialising structured data defined in a platform neutral Interface Definition Language (IDL), *proto3*. It basically takes data structures defined using the IDL and generates code in one of the supported programming languages that allows the programmer to use them internally but, more importantly, to use as messages. Protobuf focuses on performance, so the serialised messages are small as possible, which is ideal for our purposes. The data structures are called *Messages* and are defined in a proto file. gRPC, in turn, is a powerful HTTP/2 based Remote Procedure Call (RPC) framework. Instead of creating a new IDL, gRPC makes use of the *proto3* language used in protobuf, allowing for the definition of *Services*. A service has a set of RPCs methods specified using *proto3*, each one having a request and response protobuf *Message* associated. gRPC then generates code for the server and stub using the definitions.

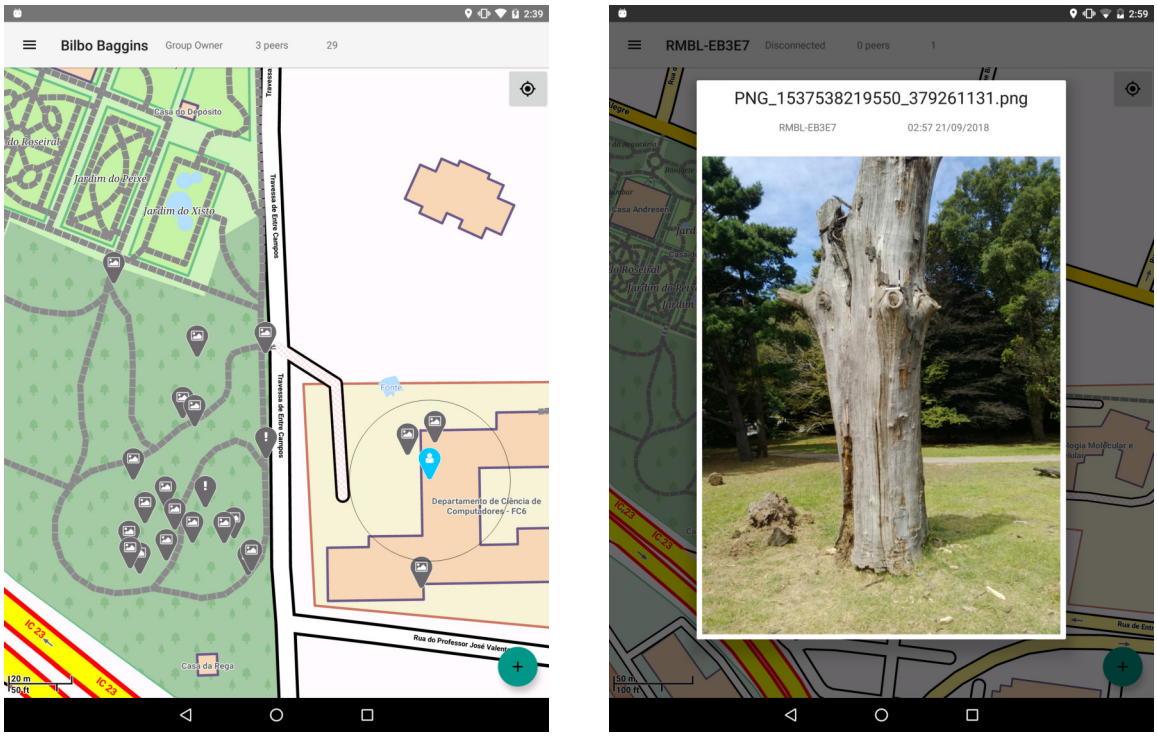
For our implementation, we were not concerned in using protobuf to generate code for different programming languages, as we decided on only using Java, but to have a well defined protocol for messages in all devices. We show the protobuf messages and services using the proto3 IDL in listing 4.1. The *Peer*, *Subscription* and *ContentMetadata* protos directly match the ones defined in figure 4.2. All messages have a header identifying the peer, followed by the contents of the message.

```
message Header {
    Peer peer = 1;
    uint64 timestamp = 2;
}

message Chunk {
    ContentMetadata metadata = 1;
    bytes data = 2;
}

message HelloRequest {
    Header header = 1;
    repeated Subscription subscription = 2;
}

message HelloResponse {
    Header header = 1;
    repeated ContentMetadata content_metadata = 2;
}

message PullRequest {
    Header header = 1;
    string content_uuid = 2;
}

message PullResponse {
    Header header = 1;
    Chunk chunk = 2;
}

service Ramble {
    rpc hello(HelloRequest) returns (HelloResponse);
    rpc pull(stream PullRequest) returns (stream PullResponse);
}
```

Listing 4.1: Services defined in proto3.

### 4.3.4 Android, Cloudlet and Cloud Apps

The Android app is the component that allows users to interact with the system. Our case study focuses on the dissemination of geotagged content between peers. To accommodate this constraint, the main view consists of a map showing markers on the map of the contents currently stored in the device, as can be seen in figure 4.3a. The user can fully interact with the map and can view the content represented by a marker by tapping




(a) Map view.

(b) Content view.

Figure 4.3: Screenshots showing the map view and a content being displayed in the Ramble app.

on it, shown in figure 4.3b. The button on the bottom right (with a plus) allows the user to generate content by taking a picture, writing a report or record audio and video. The interface was designed to be as simple as possible, hiding all the complexity of the system behind it and allowing a normal (not tech savvy) user to easily use it. The top bar indicates the username, the current P2P status (Disconnected, Peer or Group Owner) and the current version of the database. The top left icon (hamburger icon) opens a navigation drawer with the remaining views listed, consisting of a log view and a peer and content list. As the app was designed with the field experiment in mind (later described in chapter 5), the app is missing a configuration menu, where the user could adjust filters and other preferences.

The cloudlet app has the same function as the Android component. The app is the same as the Android app minus P2P discovery and the UI, with the output being displayed on the console.

### Implementation

The map view was implemented using a map rendering library called Mapsforge[4]. Mapsforge uses a specific compact map file format (compiled from other known formats using a tool) to render the map as needed. This provides a much cleaner map at the cost of computation. Mapsforge also allows the drawing of bitmaps on the map, which we used to display the markers and the precision.

The other views are simple RecyclerView lists with custom list items for each type of content being displayed.

Overall, the Android app was implemented using a single activity with multiple fragments displaying each view. The fragments are swapped around when the user requests a different view. The other software components (message and discovery services) were ran in a Service, which is a component that runs outside the Activity life cycle. The user is able to minimise the app or lock the screen while the service is running without killing it.

The other apps start the services they use and wait for them to be killed by human interaction.

[4]https://github.com/mapsforge/mapsforge

# Chapter 5

# Experiments and Results

Android, WiFi-Direct and Bluetooth have been around for a while. Despite the fact that they are used to couple smartphones and tablets with devices like headphones, fitness trackers or printers, the applications in which these technologies are used for Device-to-Device (D2D) interactions are few. There are methods to use an Android virtual machine to interact with a physical device using Bluetooth but, to our knowledge, there aren't any emulation or simulation systems for running Android applications using Peer-to-Peer (P2P) communication technologies coupled with device mobility. Static tests would not make much sense as this is a platform that takes advantage of mobility. These factors, coupled with the need to evaluate Ramble, motivated us to perform a real world test.

In section 5.1 we describe the performed experiment and the setup used in it. Section 5.2 shows the obtained results and section 5.3 contains a conclusion to this chapter.

## 5.1 Field Experiment Description

The test was performed in Porto's Botanical Garden[1]. The location was chosen mainly due to convenience, as it is located near the computer science department making logistics easier, but also because it offered a large contiguous isolated area (nearly 30000 $m^2$) for people to roam and bump into each other.

Figure 5.1 shows the setup we used in the experiment. The setup was comprised of 3 Raspberry Pi cloudlets equipped with 2 radios, one providing a WiFi Access Point (AP) (illustrated by the circumference around each cloudlet) and the other for communicating with the other cloudlets in the mesh network (illustrated by the arrows going from and to each cloudlet). The mesh network was implemented by using the ad-hoc mode and

[1] https://jardimbotanico.up.pt/

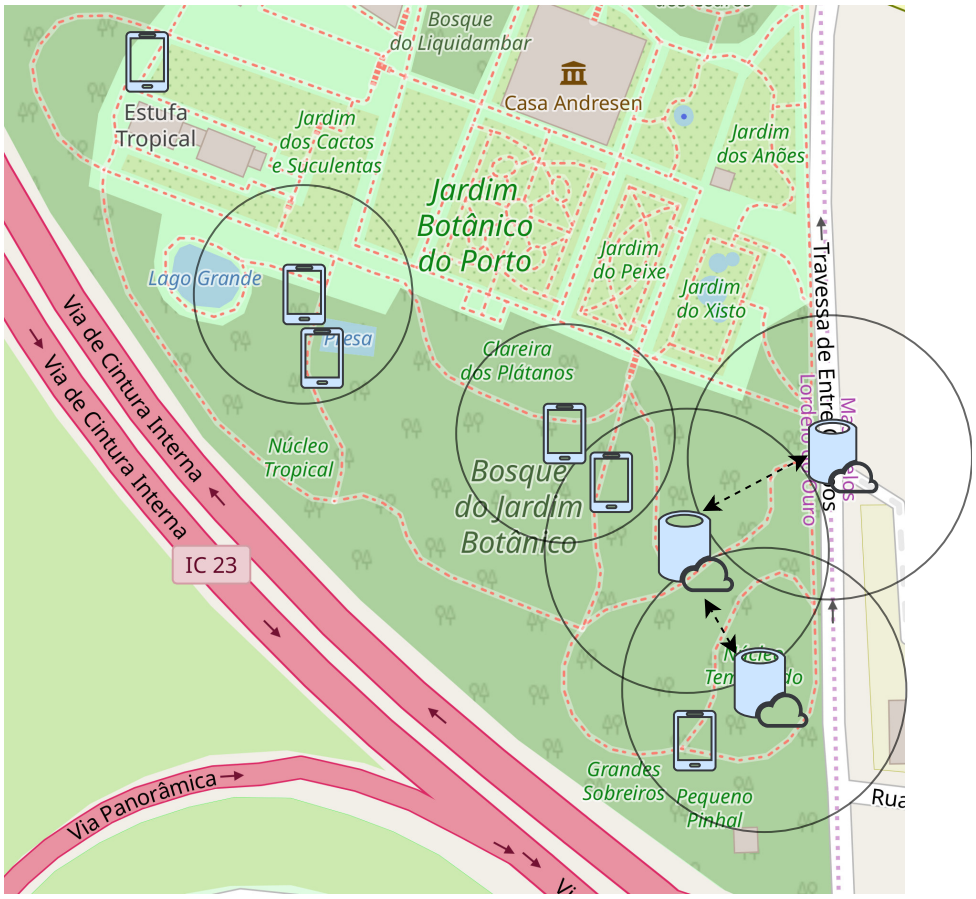



Figure 5.1: Diagram illustrating the setup used in the experiment.

the BATMAN[2] routing protocol. The remaining components of the experiment are the Android mobile devices. The Android devices we used were HTC Nexus 9 tablets running Android version 6.0, with 2GB of RAM and a dual-core 2.3GHz CPU. The cloudlets ran on Raspberry Pis 3 Model B with 1GB of SDRAM and a quad-core 1.2 GHz ARM Cortex A53 CPU.

A total of 10 people participated, each one equipped with an Android device. Prior to the experiment, a set of points of interest were collected. Users were then instructed to go to those points of interest and tag them (take a couple of pictures) if they weren't already tagged. Every user was sent from the same location (the small house in the bottom right of figure 5.1) but at different times and with alternating directions (up or left). The former was enforced to avoid users all going at the same time, which could limit the number of random encounters as they would all be in the same place at the same time, and the latter to create diversity in the mobility within the park. All devices were configured to accept all contents.

[2]https://www.open-mesh.org/

## 5.2 Results

We now proceed to examine the data collected in the experiment.

### 5.2.1 Overview

The elapsed time for the entire experiment was 3900 seconds (65 minutes). In total, 217 contents were created averaging 1.97 MB in size. These contents were exchanged 898 times between devices. Figure 5.2 shows the distribution of the contents received per device over time in a histogram type chart. In the figure it is clearly visible that the cloudlets have a smoother distribution (as they were always connected to each other) while the devices have a more irregular and random distribution. This is due to many factors concerning connection opportunities, difference between content accumulated and movement.

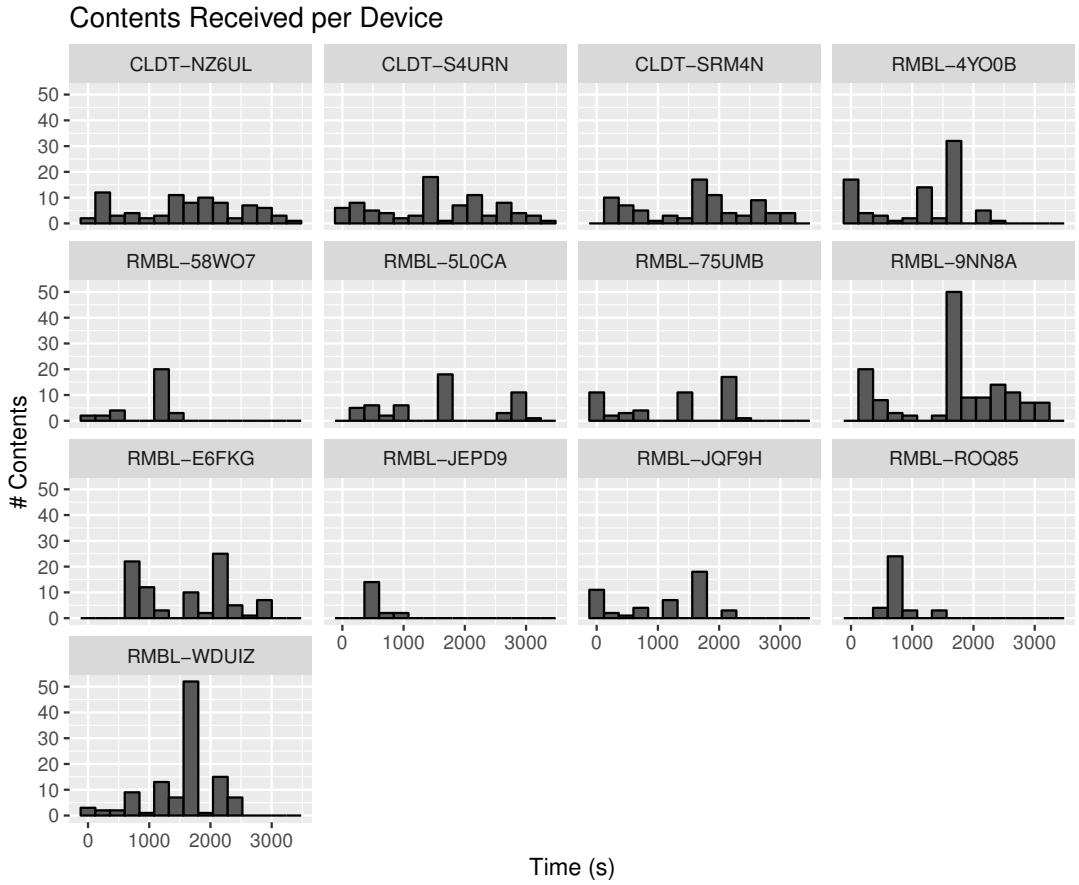


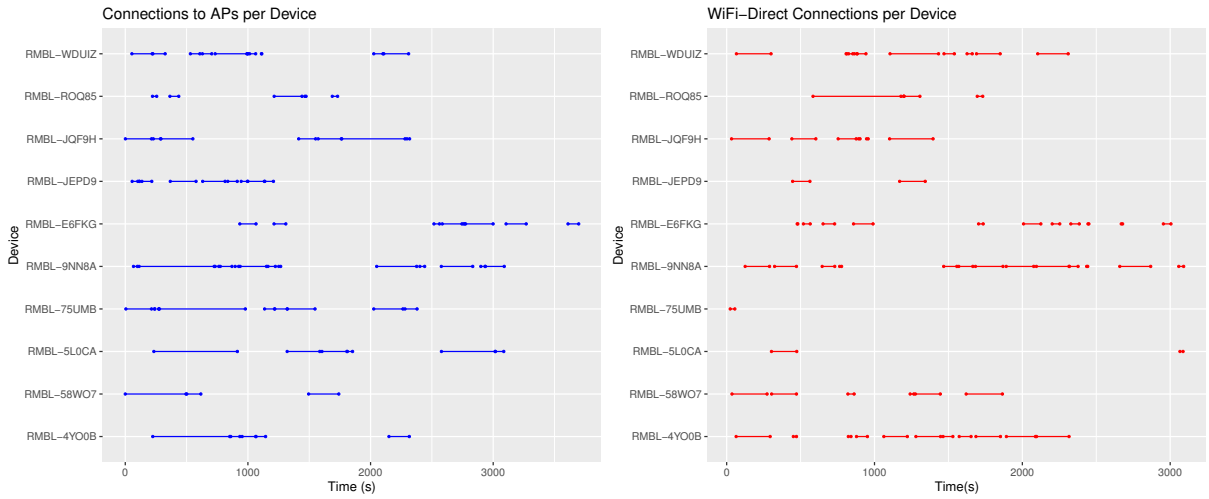

Figure 5.2: Histogram of contents received per device.


(a) GPS density. (b) RMBL-E6FKG Trace.

Figure 5.3: Density of GPS points collected in the experiment and an example of a trace made by device RMBL-E6FKG.

### 5.2.2 Movement

Regarding user movement, Global Positioning System (GPS) traces were overall very noisy and irregular in certain areas. One of the reasons for this result can be largely due to the dense vegetation in many zones of the park, interfering with the signal. To better assess this irregularity, a plot of the precision given by the Android's location manager over time would be a good solution. Unfortunately, this detail was overlooked and that information was not recorded. Even so, there are still some methods to handle noisy GPS data. We went with a simple outlier removal method that aims at reducing impossibilities, such as sudden impossible jumps and out of bounds positions. For this we considered an upper bound speed of $3m/s$ ($10.8km/h$) and replaced outliers with a plausible position between the current and the obtained position. The traces were acceptable in most devices. Figure 5.3a shows the density of GPS points collected in the experiment. The distribution is visibly circular with a higher concentration of points in the bottom right and top left corners of the park. These peaks are owed to the fact that the previously mentioned locations are more prone to user encounters. By inspection of the traces is visible that the users behaved as instructed, performing a circuit around the area, but due to the characteristics of the park or encounters with other users there was a concentration in the peak points.




(a) AP connection by device over time. (b) WiFi-Direct connection by device over time.

Figure 5.4: Plots representing active connections to APs or to other devices using WiFi-Direct over time per device.

### 5.2.3 Communication technologies used

Now we analyse contacts between devices. More specifically, we analyse contacts between mobile devices and the cloudlets, via APs, and other mobile devices, using WiFi-Direct. Important to note that in these plots, simply the use of the technology is displayed, not a connection to a peer. In figure 5.4b its often the case that one activation period had many clients.

Figure 5.4 displays two plots providing a general overview of the contacts experienced by each device. The AP connections were on average 156.4$s$ long and amounted to 30% of the total time. On the other hand they were 115.3$s$ and 19.5% on the WiFi-Direct. As can be observed in the plots, there was overlap in the usage of one or another technology. Devices under the same AP could communicate using both WiFi and WiFi-Direct. This possibility was known beforehand and wasn't explored, the system used the most recent contact established when starting the syncing process. One alternative to this behaviour could be to use both communication interfaces to split traffic and improve performance. Figure 5.5 shows boxplot of the active times.

On the other hand, the mesh network displayed an erratic behaviour from the beginning to the middle of the experiment as can be seen in Figure 5.6. We can't be sure of what caused this lack of connectivity, but we can assume that it was caused by interference, as the mesh operates in the 5GHz range, which tends to have a worse performance when facing obstacles.

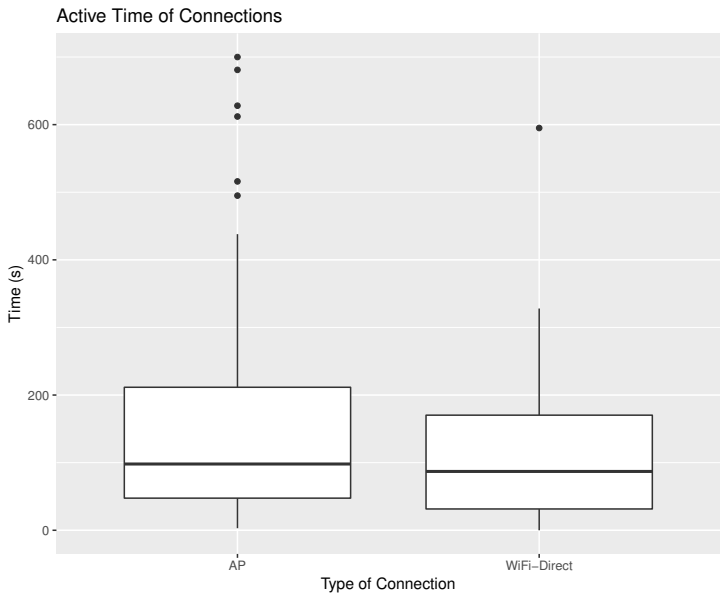



Figure 5.5: Boxplot representing the active time of AP and WiFi-Direct connections.

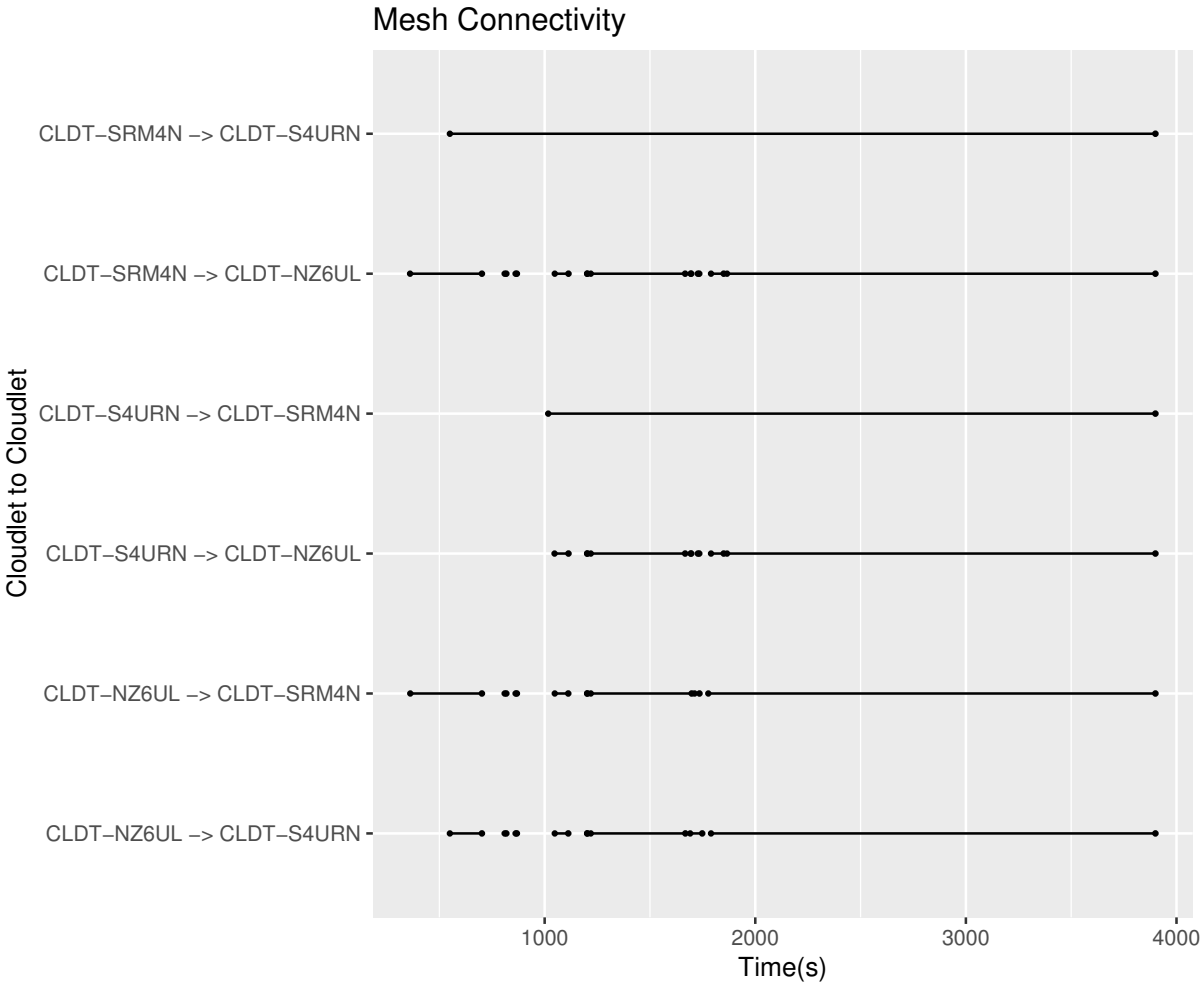



Figure 5.6: Connectivity between cloudlets using the mesh network.

### 5.2.4 Transfers

We now analyse what form of communication was used for transferring the files. Figure 5.7 displays a map in which each dot represents a transfer using APs, WiFi-Direct or both. In this map we can observe that the transfers that occurred in the area covered by APs were done while the devices were connected by all combinations of the WiFi-Direct and AP technologies, in contrast to the top-left area, where there is no coverage of APs and all transfers were done using WiFi-Direct. This presents a possible improvement point, as we expected that all transfers done in range of the cloudlets would be done using AP.

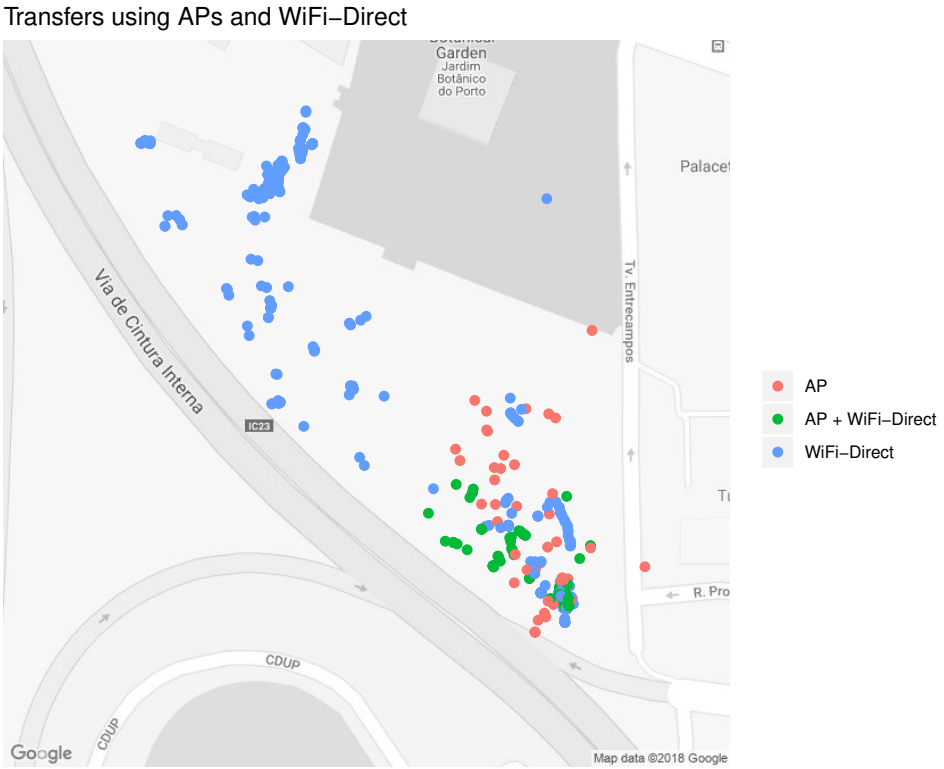



Figure 5.7: Map representing transfers using AP, WiFi-Direct or both.

### 5.2.5 Energy

When dealing with mobile devices, energy concerns are commonplace. As stated previously, the system was designed without having this concern in mind, but nonetheless we take a look at toll that our system cause on the battery of the devices. Figure 5.8 displays the battery levels of the tablets during the experiment. On average, 16% of the initial battery was used across all devices. This high usage is due to, among other factors, the screen of the devices being turned on for long periods of time, the constant usage of GPS and the WiFi-Direct scanning and connections.

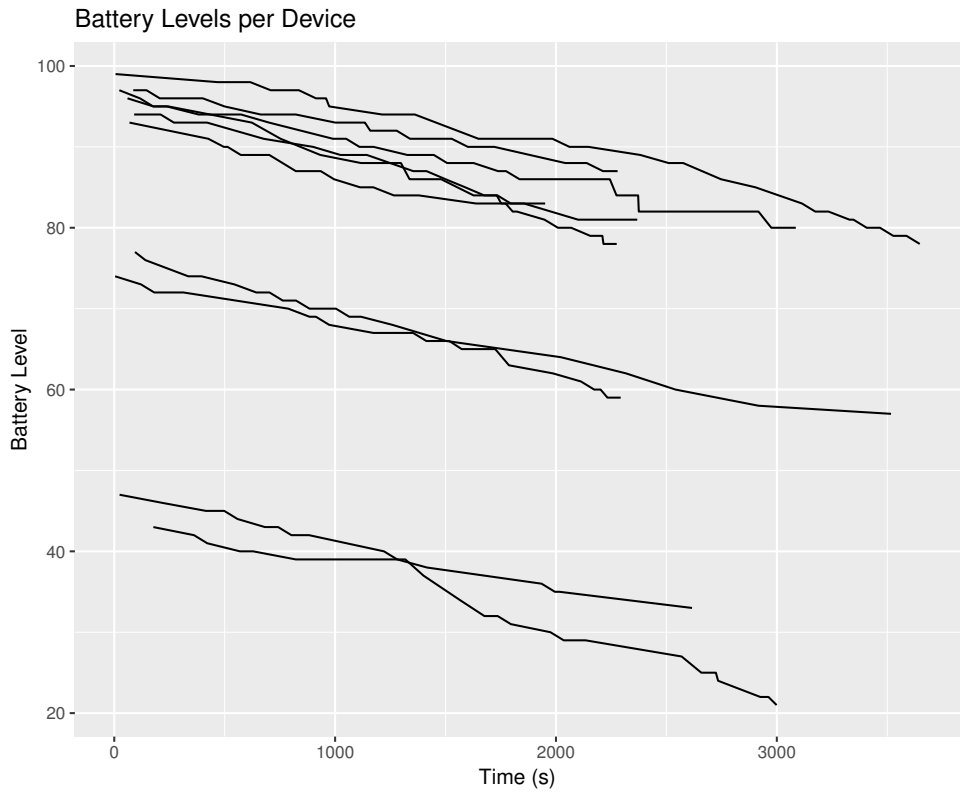



Figure 5.8: Battery usage of devices during the experiment.

## 5.3 Summary

This experiment accomplished its main goal, which was to validate the Ramble system in a real world scenario. The system was able to accomplish its main goal, which was to allow users to generate content while exchanging it with other users when they pass near each other and cloudlets.

# Chapter 6

# Conclusions and Future Work

## 6.1 Overview

In this dissertation we presented Ramble, a system for opportunistic dissemination of content for infrastructure deprived environments. We designed and developed the multiple components that make up the system in a compartmentalised way, while leaving room from improvement. We implemented a database that serves as the base of all other components, acting as a local source of truth. We implemented a discovery service general enough that it could be used in various network configurations and technologies, such as WiFi, WiFi-Direct and WiFi Ad-hoc. We designed a procedure for establishing WiFi-Direct networks between nearby mobile devices that focuses on short-term opportunistic contacts. On top of those, we then developed a message service that interact with the discovered devices with the goal of synchronising data between them. To use this, we developed applications for different physical devices, them being Android devices and cloudlets, that use the system for exchanging georeferenced contents. In particular, we developed an Android app that allows users to view and generate contents using their location. Finally we performed a real world test as proof of concept, proving that the system works and has advantages when compared with the traditional cloud computing paradigm, which would not be possible in a disconnected environment.

## 6.2 Future Work

This work was built from scratch and this was the first version, so there is obviously a lot of room for improvement.

First of all, the User Datagram Protocol (UDP) broadcast-based discovery mechanism

could be improved to multicast or replaced by already established service discovery methods, such as DNS-SD. Some networks don't allow broadcasting, so the current method could be rendered useless in such a network. The Peer-to-Peer (P2P) Discovery and Group formation procedure also has some known flaws that can cause interrupted synchronisations. This occurs because the discovery and message services are independent, and the discovery service is not aware if contents are being transferred or not. Another problem with the P2P Group Formation is the limitations of WiFi-Direct. One solution to this would be to use different communication interfaces (WiFi and Bluetooth) and alternate between a static and opportunistic mode based on sensors or number of discovered peers. That way, depending on the mobility of the user, different connection methods could be applied.

Another improvement area would be the energy concern. Constant WiFi-Direct scanning drains the battery unnecessarily, as no connections are being established. A solution would be a hybrid procedure using Bluetooth LE for discovery and WiFi-Direct for establishing connections. Although we did experiment with this, we decided not to follow that path, as there was a loss in connection time.

Finally, the database layer could be improved to use smart caching methods. A mobile device has limited memory. The filters help to limit the amount of contents a device receives. However, a device with very strict filters is limiting the spread of information. The system could be improved to prioritise the desired content, but also, based on some metrics, store content it doesn't want in order to transfer it to another peer who might.

# Bibliography


[1] Statista, "Number of smartphone users worldwide from 2014 to 2020 (in billions)," https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, [Accessed: November 2017].

[2] M. Zook, M. Graham, T. Shelton, and S. Gorman, "Volunteered geographic information and crowdsourcing disaster relief: a case study of the haitian earthquake," *World Medical & Health Policy*, vol. 2, no. 2, pp. 7–33, 2010.

[3] M. Poblet, E. García-Cuesta, and P. Casanovas, "Crowdsourcing tools for disaster management: A review of platforms and methods," in *AI Approaches to the Complexity of Legal Systems*. Springer, 2014, pp. 261–274.

[4] "Scipionus - Hurricane Information Maps," https://gregstoll.dyndns.org/scipionus/, [Accessed: January 2018].

[5] H. Yamamura, K. Kaneda, and Y. Mizobata, "Communication problems after the great east japan earthquake of 2011," *Disaster medicine and public health preparedness*, vol. 8, no. 4, pp. 293–296, 2014.

[6] P. M. Mell and T. Grance, "SP 800-145. The NIST Definition of Cloud Computing," NIST, Gaithersburg, MD, United States, Tech. Rep., 2011.

[7] "Gabriel - Wearable Cognitive Assistance using Cloudlets," http://gabriel.cs.cmu.edu/index.html, [Accessed: January 2018].

[8] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, 2013.

[9] W.-F. Alliance, "Wi-fi peer-to-peer (p2p) technical specification v1. 7," *https://www.wi-fi.org/file/wi-fi-peer-to-peer-p2p-technical-specification-v17*, 2016.

[10] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with wi-fi direct: overview and experimentation," *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[11] P. M. P. Silva, J. Rodrigues, J. Silva, R. Martins, L. Lopes, and F. Silva, "Using edge-clouds to reduce load on traditional wifi infrastructures and improve quality of experience," in *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*. IEEE, 2017, pp. 61–67.

[12] J. Rodrigues, E. R. B. Marques, J. Silva, L. Lopes, and F. Silva, "Video dissemination in untethered edge-clouds: a case study," in *Proc. 18th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, ser. DAIS'18. Springer, 2018, pp. 137–152.

[13] J. Rodrigues, E. R. B. Marques, L. M. B. Lopes, and F. Silva, "Towards a middleware for mobile edge-cloud applications," in *Proceedings of the 2Nd Workshop on Middleware for Edge Clouds & Cloudlets*, ser. MECC '17. New York, NY, USA: ACM, 2017, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/3152360.3152361

[14] T. Freitas, "Crowdsourcing photos in edge-clouds with panoptic," Master's thesis, Faculdade de Ciências da Universidade do Porto, 2017.

[15] M. Silva, "Hyraxmsg: uma aplicação de "messaging" para "mobile edge clouds"," Faculdade de Ciências da Universidade do Porto, Tech. Rep., 2017.

[16] "The Serval Project," http://www.servalproject.org/, [Accessed: December 2017].

[17] "FireChat," https://www.opengarden.com/firechat.html, [Accessed: December 2017].

[18] "MeshKit," https://www.opengarden.com/meshkit.html, [Accessed: January 2018].

[19] "Commotion Wireless," https://commotionwireless.net/, [Accessed: December 2017].

[20] "goTenna - Text & GPS on your phone, even without service," https://gotenna.com/, [Accessed: January 2018].

[21] "p2pkit by Uepaa," https://p2pkit.io/, [Accessed: January 2018].

[22] "AllJoyn," https://openconnectivity.org/developer/reference-implementation/alljoyn, [Accessed: January 2018].

[23] "FrontlineSMS," https://www.frontlinesms.com/, [Accessed: January 2018].

[24] "OpenStreetMap," https://www.openstreetmap.org/, [Accessed: January 2018].

[25] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[26] "Ushahidi," https://www.ushahidi.com/, [Accessed: January 2018].

[27] O. Okolloh, "Ushahidi, or 'testimony': Web 2.0 tools for crowdsourcing crisis information," *Participatory learning and action*, vol. 59, no. 1, pp. 65–70, 2009.

[28] "Zello," https://zello.com/, [Accessed: January 2018].

[29] W. Post, "Hurricane Irma just made a digital walkie-talkie the No. 1 app online," https://www.washingtonpost.com/news/innovations/wp/2017/09/06/hurricane-irma-just-made-a-digital-walkie-talkie-the-no-1-app-online/, [Accessed: January 2018].